Моделирование и анализ информационных систем. Т. 24, № 2 (2017), с. 215–226
Modeling and Analysis of Information Systems. Vol. 24, No 2 (2017), pp. 215–226

DOI: 10.18255/1818-1015-2017-2-215-226

УДК 004.056.3

Дедубликация в системе резервного копирования с хранением информации в базе данных

Таранин С. М.

получена 18 сентября 2016

Аннотация. Профилактика потери данных с цифровых носителей включает такой процесс, как резервное копирование. Он может проводиться вручную простым копированием данных на внешние носители или автоматизированно по расписанию с помощью специальных программных средств. Существуют системы удаленного резервного копирования, когда данные сохраняются по сети в удаленное хранилище. Такие системы являются многопользовательскими и обрабатывают большие объемы данных. В общем хранилище могут встретиться файлы, содержащие одинаковые фрагменты. Для исключения повторяющихся данных применяется механизм дедубликации (англ. de-duplication). Он представляет собой метод сжатия информации, когда поиск копий производится по всему массиву данных, а не в пределах одного файла. Главным преимуществом использования данной технологии является существенная экономия дискового пространства. Однако механизм исключения повторяющихся данных может существенно снизить скорость сохранения и восстановления информации. Настоящая статья посвящена проблеме реализации такого механизма в системе резервного копирования с хранением информации в реляционной базе данных. В данной работе рассматривается пример реализации такой системы, работающей в двух режимах: с дедубликацией данных и без нее. В статье приведен пример схемы классов для разработки клиентской части приложения, а также описание таблиц и связей между ними в базе данных, что относится к серверной части. Далее автор предлагает алгоритм сохранения данных с дедубликацией, а также приводит результаты сравнительных тестов скорости работы алгоритмов сохранения и восстановления информации при работе с реляционными системами управления базами данных разных производителей.

Ключевые слова: файл, данные, резервное копирование, дедубликация, база данных

Для цитирования: Таранин С. М., "Дедубликация в системе резервного копирования с хранением информации в базе данных", *Моделирование и анализ информационных систем*, **24**:2 (2017), 215–226.

Об авторах:
Таранин Сергей Максимович, orcid.org/0000-0001-8117-7358, аспирант,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: staranin0208@yandex.ru

Введение

Одной из угроз целостности данных на автоматизированном рабочем месте (АРМ) является выход из строя цифрового энергонезависимого устройства хранения информации. Примерами таких устройств являются жесткие диски (англ. hard disk

drive HDD), твердотельные накопители (англ. solid-state drive, SSD), а также гибридные устройства (англ. solid-state hybrid drive, SSHD), представляющие собой компромиссное решение между стоимостью первых и производительностью последних. Нарушение целостности данных на цифровых носителях может произойти вследствие различных причин. Главным фактором, ограничивающим длительность хранения информации на цифровых носителях, является срок их службы. Он может быть разным в зависимости от производителя, однако даже у самых качественных и дорогих устройств срок службы составляет в среднем пять лет.

Профилактика потери данных с цифровых носителей включает такой процесс, как резервное копирование [2]. Он может проводиться вручную простым копированием данных на внешние носители или автоматически по расписанию с помощью специальных программных средств. Существуют системы удаленного резервного копирования, когда данные сохраняются по сети в удаленное хранилище. Такие системы являются многопользовательскими и обрабатывают большие объемы данных. В общем хранилище могут встретиться файлы, содержащие одинаковые фрагменты. Для исключения повторяющихся данных применяется механизм дедубликации (англ. de-duplication)[7,8]. Он представляет собой метод сжатия информации, когда поиск копий производится по всему массиву данных, а не в пределах одного файла. Главным преимуществом использования данной технологии является существенная экономия дискового пространства. Однако при этом снижается производительность системы резервного копирования.

Настоящая статья посвящена проблеме реализации механизма исключения дублирования данных в системе резервного копирования с хранением информации в базе данных (БД). В рамках данной работы рассмотрено использование реляционных систем управления базами данных (СУБД), что связано с их широкой распространенностью и встроенными механизмами обеспечения целостности [10,11,12]. В ходе работы будет предложена схема реализации клиентской и серверной части системы [9], алгоритм сохранения данных с дедубликацией. Также будут приведены и проанализированы результаты сравнительных тестов производительности системы с включенным и выключенным механизмом исключения дублирования данных при работе с СУБД разных производителей.

1. Реализация системы резервного копирования

В работе [1] описан подход к реализации системы резервного копирования с хранением в БД без дедубликации. В таком случае для хранения пользовательских данных на сервере предлагается завести в БД две таблицы $Model$ и $Data$. В первой хранится информация об объекте файловой системы, а во второй – данные этого объекта, разбитые на блоки фиксированной длины. Связь между таблицами $Model$ и $Data$ «один ко многим» соответственно. Таким образом, на каждый блок данных указывает только один файл.

Для исключения повторяющихся данных в таблице $Data$, ее связь с таблицей $Model$ должна быть «многие ко многим», когда на один блок данных могут ссылаться несколько файлов. Для организации такой связи предлагается ввести дополнительную таблицу $Link$.

Описание таблицы *Model* остается как в первоначальном варианте [1], а из таблицы *Data* удалено поле $FILEID$. Таблица *Link* содержит следующие поля:

1. ID Первичный ключ.
2. FILEID Внешний ключ. Идентификатор записи в таблице *Model*.
3. DATAID Внешний ключ. Идентификатор записи в таблице *Data*.
4. ORD Порядковый номер блока с идентификатором $DATAID$ в файле с идентификатором $FILEID$.

Порядок блоков внутри файла обеспечивается за счет значений ORD. Следующий SQL-запрос по идентификатору записи таблицы *Model* вернет последовательность блоков в том порядке, в котором они идут внутри файла:

```
select Data.BLOCK
   from Link
   inner join Data
   on Link.FILEID = [идентификатор] and Link.DATAID = Data.ID
order by Link.ORD;
```

Клиентская часть системы резервного копирования представляет собой приложение, которое должно уметь читать и записывать содержимое пользовательских файлов, взаимодействовать с СУБД путем выполнения *SQL*-запросов, а также шифровать данные при необходимости. Выделим основные компоненты клиентского приложения:

1. Ядро. Содержит реализацию классов для представления пользовательских данных в виде набора записей таблицы, а также описание интерфейсов для взаимодействия с СУБД и файловой системой пользователя.
2. Контроллер для взаимодействия с СУБД. Реализует интерфейс для взаимодействия с СУБД. Таких контроллеров может быть несколько, по количеству поддерживаемых СУБД разных производителей. Общую часть, которая одинакова для всех СУБД, можно вынести в базовый класс.
3. Контроллер для работы с файловой системой [15]. Реализует интерфейс для взаимодействия с файловой системой пользователя, реализацию алгоритмов сохранения (с дедубликацией и без нее), восстановления данных, а также сканирования директории на наличие изменений.
4. Криптопровайдер. Реализует алгоритмы шифрования и хеширования данных.
5. Модель копируемой директории. Является представлением набора записей таблицы *Model* на стороне клиента.

На рисунке 1 представлен пример схемы классов для реализации клиентского приложения.

Ядро системы составляют классы *Record* и *RecordSet*. Класс *Record* необходим для представления данных на стороне клиента в виде записи, а *RecordSet* – в виде упорядоченного набора записей.

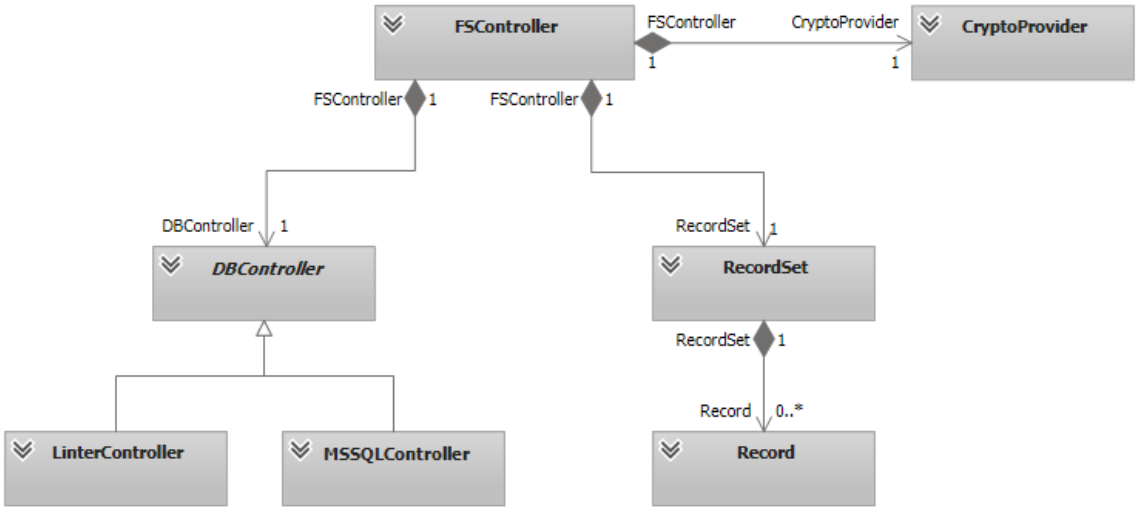

Рис. 1: Схема классов клиентского приложения

Fig. 1. The classes schem of client application

Классы *MSSQLController* и *LinterController* представляют собой реализацию контроллеров для взаимодействия с СУБД MSSQL и Linter соответственно. При необходимости можно расширять список поддерживаемых СУБД, реализуя подобные классы по некоторому общему интерфейсу. Такой интерфейс может включать функции сохранения, обновления, удаления и чтения записей таблиц по идентификатору, функции удаления и создания таблиц, функцию выборки записей по значению полей и другие. Абстрактный класс *DBController* содержит описание общего интерфейса взаимодействия с СУБД и функции, реализация которых одинакова для всех поддерживаемых систем. Например, функция удаления таблицы по имени для всех СУБД выполняет *SQL*-запрос вида:

```
drop table [имя_таблицы];
```

Класс *CryptoProvider* содержит реализацию алгоритмов шифрования хеширования данных.

Класс *FSController* представляет собой реализацию контроллера для работы с файловой системой пользователя. Он содержит актуальную копию таблицы *Model* в виде объекта типа *RecordSet*, экземпляр класса контроллера для взаимодействия с СУБД и экземпляр класса криптопровайдера.

1.1. Сохранение и восстановление данных

Предлагается следующий рекурсивный алгоритм сохранения данных с дедубликацией:

1 Для каждого файла из текущей директории:

1.1 Создаем запись для таблицы *Model* и сохраняем ее *insert*-запросом к БД.

1.2 Получаем идентификатор сохраненной записи.

1.3 Сохраняем запись в таблице *Model* на стороне клиента.

1.4 Читаем файл с диска блоками, формируя массив записей для сохранения в таблицу *Data*.

1.5 Для каждого блока сохраняемого файла:

1.5.1 Считаем хеш-код блока

1.5.2 Если в таблице *Data* нет блока с таким хеш-кодом, сохраняем новую запись в таблицу *Data* *insert*-запросом к БД.

1.5.3 Создаем запись для таблицы *Link* и сохраняем ее *insert*-запросом к БД.

2 Для каждой папки из текущей директории:

2.1 Формируем запись для таблицы *Model* и сохраняем ее *insert*-запросом к БД.

2.2 Запускаем данный алгоритм для файлов в этой директории.

3 Считаем хэш-код для текущей директории и сохраняем его в поле *HASH* соответствующей записи таблицы *Model* *update*-запросом.

Как и в случае без дедубликации, выведем зависимость количества запросов к БД от размера блока при полном сохранении всех данных. Пусть n – количество файлов в текущей директории, m – количество папок в текущей директории, а sgm – длина блока. Число блоков файла длины l равно:

$$s = floor\left(\frac{l + sgm - 1}{sgm}\right),$$

где $floor(x)$ – функция нахождения ближайшего целого, не превышающего x. Тогда количество запросов к БД при сохранении содержимого текущей директории можно выразить следующей формулой:

$$F(m, n) = \sum_{i=1}^{n} (2s_i + \epsilon_i + 1) + m + 1,$$

где $sgm < l_i$, l_i – длина i-го файла в текущей директории, s_i – число блоков i-го файла, а $0 \leq \epsilon_i \leq s_i$ – количество сохраненных блоков.

Для каждого файла выполняется s_i *select*-запросов для определения наличия блока с заданным хешем в таблице *Data* и столько же *insert*-запросов в таблицу *Link*, ϵ_i *insert*-запросов для записи блоков в таблицу *Data*, один запрос для записи информации о файле в таблицу *Model*, m запросов для записи информации о папках в таблицу *Model* и еще один запрос для обновления записи в таблице *Model* (сохранение хеша текущей директории). ϵ будет равен нулю в случае, если директория содержит только копии уже сохраненных файлов, и примет значение s_i, если в директории отсутствуют копии сохраненных файлов.

Количество запросов к БД при сохранении содержимого директории и всех ее вложенных папок можно представить следующей формулой:

$$F_k = \sum_{i=1}^{n_k} (2s_{k_i} + \epsilon_{k_i}) + \sum_{j=1}^{m_k} F(m_{k_j}, n_{k_j}) + m + 1,$$

где k – идентификатор текущей директории.

В случае сохранения данных без дедубликации, формула имеет вид:

$$F_k = \sum_{i=1}^{n_k} (s_{k_i}) + \sum_{j=1}^{m_k} F(m_{k_j}, n_{k_j}) + m + 1.$$

Таким образом, за счет дедубликации, скорость сохранения уменьшится из-за дополнительных $\sum_{i=1}^{n_k} (s_{k_i} + \epsilon_{k_i})$ запросов.

Алгоритм восстановления данных остается таким же, как и в случае без дедубликации [1]. Меняется только текст запроса, который возвращает содержимое файла. Соответственно дедубликация практически не влияет на скорость восстановления данных.

1.2. Тестирование

Посмотрим, как это работает на практике. Тестовый стенд представляет собой два персональных компьютера (ПК), объединенных в сеть. Один из них играет роль сервера с установленной СУБД (процессор: Intel Core i5-2500K 3,3 GHz L3 6МБ, память 8 GB). Второй ПК представляет собой клиентскую часть системы, которая содержит пользовательские данные (процессор: Intel Celeron 2,8 GHz L2 256КБ, память: 3 GB). Жесткие диски обоих ПК отформатированы под файловую систему NTFS(3,15). Размер кластера составляет 4 КБ.

Система тестируется с применением СУБД Linter 6.0.18.9 Demo и MSSQL 2008 R2 с настройками по умолчанию [13,14].

СУБД MSSQL содержит компонент управления буфером, состоящий из двух механизмов: диспетчер буферов для доступа и обновления страниц базы данных, а также буферный кэш (известный как буферный пул) для сокращения операций ввода-вывода файла базы данных. На тестовом стенде заданы следующие настройки буферного кэша: количество доступной физической памяти (bpool_commit_target) 530358 КБ, объем физической памяти в диспетчере (bpool_committed) 8945 КБ.

СУБД Linter для своей работы использует так называемые очереди таблиц, файлов, колонок таблиц и пользователей. Размеры очередей задаются количеством элементов, размеры которых в разных очередях колеблются в среднем от 50 до 1500 байт, согласно документации. Тестовые данные были получены при следующих настройках: кэш файлов – 20 элементов, кэш таблиц – 100 элементов, кэш колонок – 500 элементов, кэш каналов – 100 элементов.

На стороне клиента запускается приложение, которое обрабатывает данные пользователя и отправляет их на сервер без предварительного шифрования. После того как все данные успешно сохранены, клиентское приложение запрашивает их обратно и сохраняет их на стороне клиента в другой директории. В качестве тестовых

данных были выбраны файлы разного размера. Для первых двух тестов было отобрано 76 файлов размером 0,6 МБ и столько же файлов по 3 МБ. В трёх других тестах происходит сохранение одного файла размером 22 МБ, 162 МБ и 1,36 ГБ. Для каждого теста все файлы расположены в одной директории без вложенных папок.

Наша задача заключается в выборе оптимального размера блока для файлов разного размера при включенном механизме исключения дублирования данных. Скорость обработки данных при использовании разных СУБД сильно отличается, однако характер влияния дедубликации на скорость сохранения в обоих случаях идентичен. Чем больше размер блока, тем меньше дедубликация влияет на скорость сохранения данных. С дедубликацией, при увеличении размера блока, скорость работы сохранения алгоритма растет быстрее до определенного момента.

Минимальный размер блока, 4 КБ, равен размеру кластера жесткого диска, отформатированного под файловую систему NTFS, а максимальный – 1 МБ. Выбор максимального размера блока связан с тем, что объекты размером 1 МБ и более эффективнее хранить в файловой системе, чем в БД, поскольку NTFS лучше справляется с фрагментацией таких объектов. Для хранения данных размером 256–1024 КБ ни файловая система, ни БД не дают явного преимущества. Объекты менее 256 КБ эффективнее хранить в БД [5].

Рассмотрим результаты первого теста (рисунок 2). Для СУБД MSSQL скорость сохранения данных растет при увеличении длины блока до 16 КБ, а потом не изменяется. Скорость восстановления начинает падать сразу с 4 КБ. Для СУБД Linter при длине блока более 32 КБ скорость сохранения данных растет медленнее, а скорость восстановления начинает увеличиваться. Таким образом, оптимальный размер блока составляет 32 КБ для Linter и 16 КБ для MSSQL. При этом файлы поделятся примерно на 19 и 38 частей соответственно.

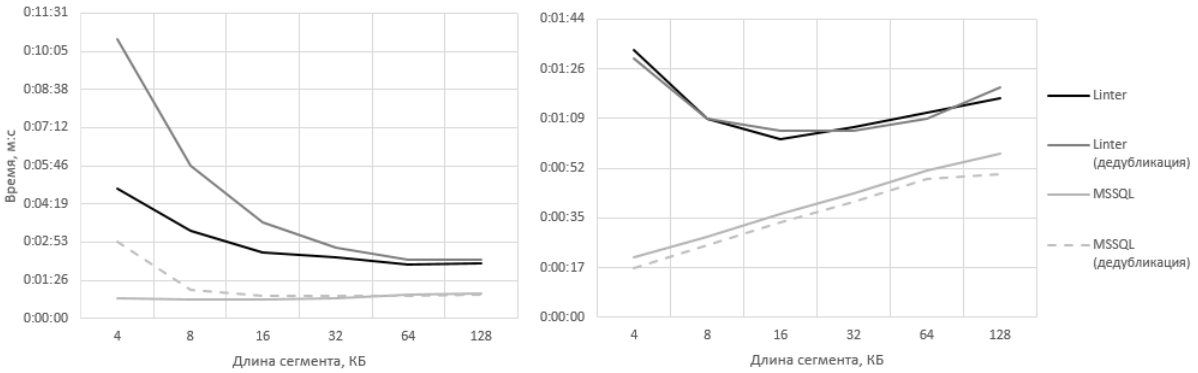

Рис. 2: Сохранение (слева) и восстановление (справа) 76 файлов по 0,6 МБ

Fig. 2. The saving (left) and recovery (right) of 76 files by 0,6 MB

Рассмотрим результаты второго теста (рисунок 3). Для СУБД MSSQL при увеличении длины блока более 16 КБ скорость сохранения данных растет медленнее, а скорость восстановления, так же как и в первом тесте, начинает падать сразу с 4 КБ. Для СУБД Linter увеличение блока больше 64 КБ не влияет на скорость восстановления, а скорость сохранения растет медленнее. Таким образом, оптимальный

размер блока может составлять 64 КБ для Linter и 16 КБ для MSSQL. При этом файлы будут разбиты примерно на 47 и 187 частей соответственно.

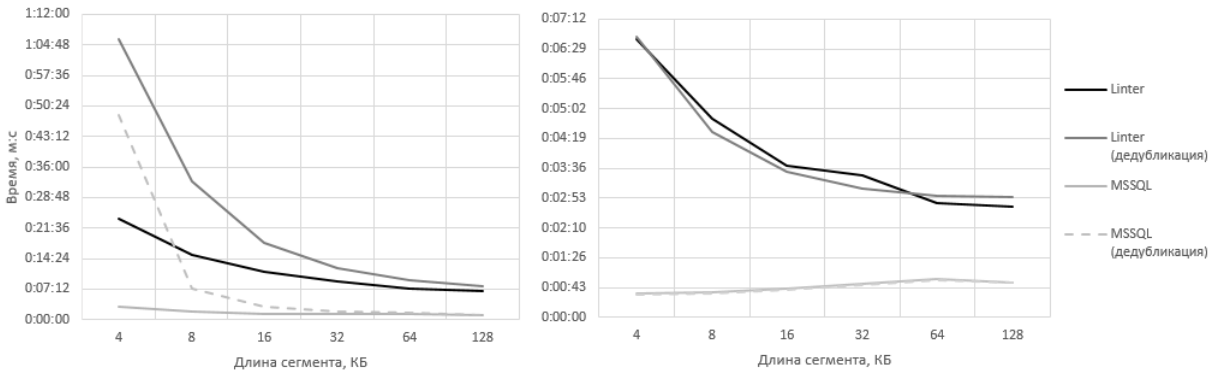

Рис. 3: Сохранение (слева) и восстановление (справа) 76 файлов по 3 МБ

Fig. 3. The saving (left) and recovery (right) of 76 files by 3 MB

Для СУБД MSSQL, при сохранении файла размером 22 МБ (рисунок 4), скорость сохранения растет быстро, а скорость восстановления не меняется при увеличении длины блока. Для Linter при увеличении блока больше 64 КБ скорость сохранения растет медленно, а скорость восстановления не меняется. Таким образом, оптимальный размер блока может составлять 64 КБ для Linter и 128 КБ для MSSQL. При этом файл будет разбит примерно на 344 и 688 частей соответственно.

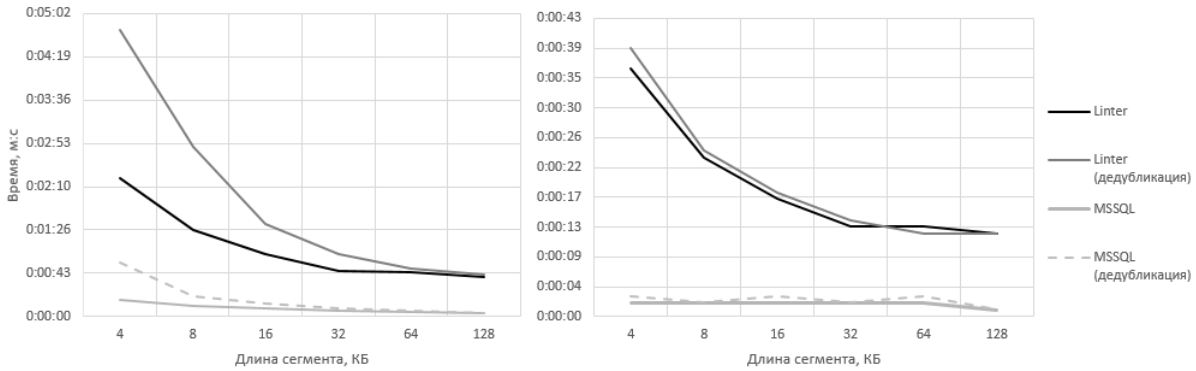

Рис. 4: Сохранение (слева) и восстановление (справа) файла, 22 МБ

Fig. 4. The saving (left) and recovery (right) of file, 22 MB

Для СУБД MSSQL, при сохранении файла размером 162 МБ (рисунок 5), скорость сохранения и восстановления растет быстро и равномерно для блоков больше 64 КБ. Для Linter увеличение длины блока более 256 КБ незначительно влияет на скорость сохранения данных, а увеличение более чем на 128 КБ незначительно влияет на скорость восстановления. Таким образом, оптимальный размер блока для Linter может составить 256 КБ, а для MSSQL – 1 МБ. При этом файл будет разбит примерно на 633 и 158 частей соответственно.

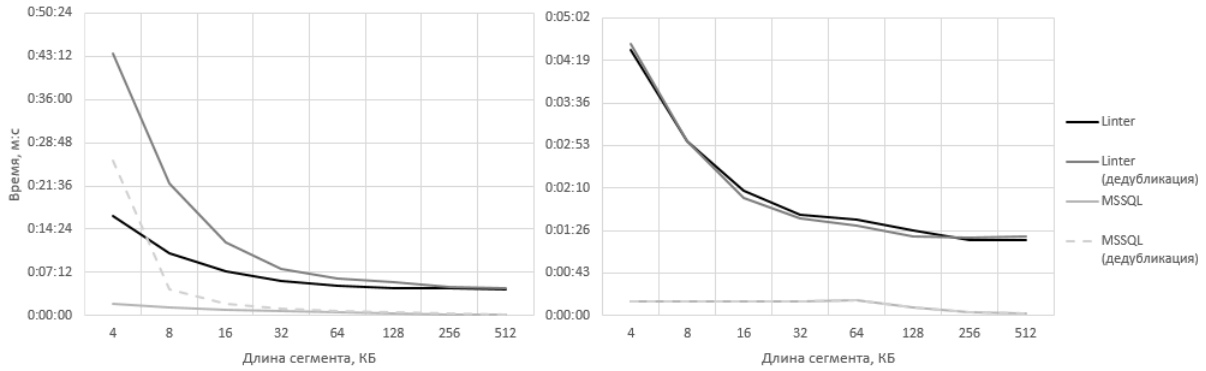

Рис. 5: Сохранение данных (1 файл, 162 МБ)

Fig. 5. The data saving (1 file, 162 MB)

В последнем тесте (рисунок 6) для СУБД Linter характер зависимости не изменился. Для MSSQL увеличение длины блока более 1 МБ незначительно влияет на скорость сохранения данных, а скорость восстановления быстро растет для блоков, длина которых больше 64 КБ. Однако для более эффективного хранения необходимо оставить длину блока на уровне 1 МБ. Таким образом, оптимальный размер блока может составлять 256 КБ для Linter и 1 МБ для MSSQL. При этом файл будет разбит примерно на 5312 и 1328 частей соответственно.

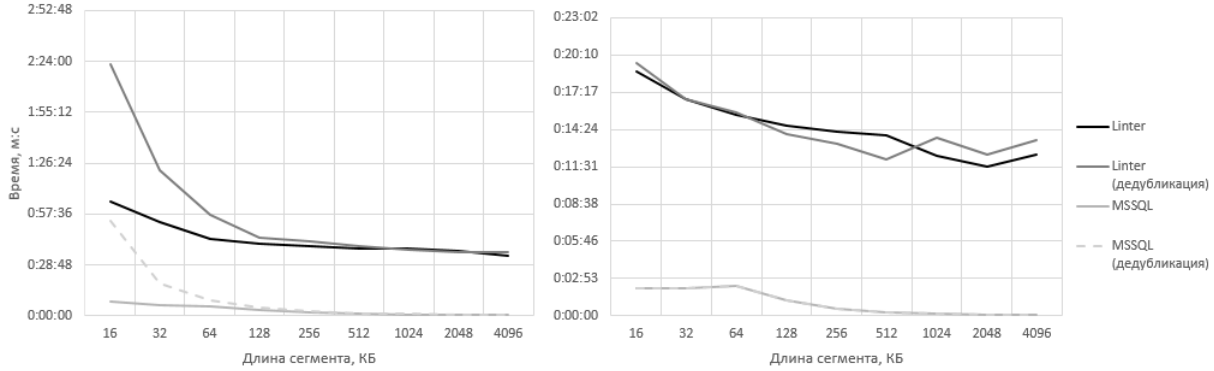

Рис. 6: Сохранение данных (1 файл, 1,36 ГБ)

Fig. 6. The data saving (1 file, 1,36 GB)

Таким образом, чтобы добиться наибольшей эффективности работы системы резервного копирования с дедубликацией, необходимо менять размер блока в зависимости от размера файла и СУБД, которая установлена на сервере (адаптивная дедубликация) [4]. Результаты тестов приведены в таблице 1.

Таблица 1. Результаты тестов

Table 1. The tests results

Тесты Tests		СУБД DBMS	
		MSSQL	Linter
Номер Number	Данные количество файлов / размер файла, МБ Data the files count / the size of file, MB	Оптимальный размер блока, КБ The optimal size of block, KB	
1	70 / 0.6	16	32
2	76 / 3	16	64
3	1 / 22	128	64
4	1 / 162	1024	256
5	1 / 1360	1024	256

Заключение

В статье был предложен подход к реализации механизма дедубликации в системе резервного копирования с хранением информации в базе данных. Архитектура такой системы позволяет внедрить данную технологию без существенного изменения программного кода. В зависимости от сетевой инфраструктуры, в которую устанавливается данная система, механизм исключения дублирования данных может быть выключен для достижения максимальной производительности. Если дедубликация необходима, то для повышения эффективности работы системы следует менять мелкость разбиения файлов в зависимости от их размера, а также от СУБД, которая обрабатывает данные на стороне сервера. В статье были приведены результаты соответствующих тестов.

Направлением дальнейшей работы является исследование поведения клиентской и серверной части предлагаемой системы при изменении файлов на клиенте, а также сетевой нагрузки при этом.

Список литературы / References

[1] Таранин С. М., “Резервное копирование с хранением в базе данных”, *Моделирование и анализ информационных систем*, **23**:4 (2016), 479–491; [Taranin S. M., “Backup with Storage in a Database”, *Modeling and Analysis of Information Systems*, **23**:4 (2016), 479–491, (in Russian).]

[2] Казаков В. Г., Федосин С. А., “Технологии и алгоритмы резервного копирования”, *Всероссийский конкурсный отбор обзорно-аналитических статей по приоритетному направлению «Информационно-телекоммуникационные системы»*, 2008, 1–49; [Kazakov V. G., Fedosin S. A., “Technologii i algoritmi reservnogo kopirovania”, *Vserossiyskiy konkursniy otbor obzorno-analiticheskih statey po prioritetnomu napravleniu "Informacionno-telekommunikacionnie sistemi"*, 2008, 1–49, (in Russian).]

[3] Medeiros J., “NTFS Forensics: A Programmers View of Raw Filesystem Data Extraction”, *Grayscale Research*, 2008, 1–27.

[4] Казаков В. Г., Федосин С. А., Плотникова Н. П., “Способ адаптивной дедупликации с применением многоуровневого индекса размещения копируемых блоков данных”, *Фундаментальные исследования*, 2013, № 8, 1322–1325; [Kazakov V. G., Fedosin S. A., Plotnikova N. P., “Method of adaptive dedublication with multilevel block indexing”, *Fundamental research*, 2013, № 8, 1322–1325].

[5] Sears R., Catharine van Ingen, Gray J., To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem? *Technical Report MSR-TR-2006-45*, 2006, 1–11.

[6] Zhu N., Chiueh T., “Portable and Efficient Continuous Data Protection for Network File Servers”, *Stony Brook University*, 2007, 1–17.

[7] Meyer D. T., Bolosky W. J., “A Study of Practical Deduplication”, *ACM Transactions on Storage*, **7**:4 (2012), 1–13.

[8] Storer M. W., Greenan K., Long D. D. E., Miller E. L., “Secure Data Deduplication”, *Proceedings of the 4th ACM international workshop on Storage security and survivability*, 2008, 1–10.

[9] Renzel K., Keller W., “Client/Server Architectures for Business Information Systems”, *A Pattern Language*, 1997, 1–25.

[10] Дейт К. Дж., *Введение в системы баз данных*, **8**, Вильямс, 2005; In English: Date C. J., *An Introduction to Database Systems*, **8**, Pearson Education, Inc., 2004.

[11] Грофф Д., Вайнберг П., Оппель Э., *SQL: полное руководство*, **3**, Вильямс, 2015; In English: Groff J., Weinberg P., Oppel A., *SQL The Complete Reference*, **3**, The McGraw-Hill Companies, 2010.

[12] Дейт К. Дж., *SQL и реляционная теория. Как грамотно писать код на SQL*, Символ-Плюс, 2010; In English: Date C. J., *SQL and Relational Theory. How to Write Accurate SQL Code*, O’Reilly Media Inc., 2009.

[13] Mistry R., Misner S., *Introducing Microsoft SQL Server 2008 R2*, Microsoft Press, 2010.

[14] Максимов В., Козленко Л. А., Маркин С. П., Бойченко И. А., “Защищенная реляционная СУБД Линтер”, *Открытые системы. СУБД*, 1999, № 11–12; [Maksimov V., Kozlenko L. A., Markin C. P., Bojchenko I. A., “Zashchishchennaya relyacionnaya SUBD Linter”, *Otkrytye sistemy. SUBD*, 1999, № 11–12, (in Russian).]

[15] Таненбаум Э., Бос Х., *Современные операционные системы*, **4**, Питер, 2015; In English: Tanenbaum A. S., Bos H., *Modern Operating Systems*, **4**, Pearson Education, Inc., 2015.

Taranin S. M., "De-duplication on the Backup System with Information Storage in a Database", *Modeling and Analysis of Information Systems*, **24**:2 (2017), 215–226.

DOI: 10.18255/1818-1015-2017-2-215-226

Abstract. Prevention of data loss from digital media includes such a process as a backup. It can be done manually by copying data to external media or automated on a schedule by using special software. There are the remote backup systems, when data are saved over the network to the remote repository. Such systems are multi-user and they process large amounts of data. Shared storage can meet files containing the same fragments. The elimination of repeated data is based on the mechanism of de-duplication. It is a method of information compression, when the search of copies is performed in the entire dataset rather than within a single file. The main advantage of using this technology is a significant saving of disk space. However, the mechanism of eliminating repetitive data can significantly reduce the speed of saving and restoring information. This article is devoted to the problem of implementing such a mechanism in the backup system with information storage in a relational database. In this paper we consider an example of implementation of such a system working in two modes: with the de-duplication of data and without it. The article illustrates a class diagram for the development of a client part of application as well as the description of tables and relationships between them in a database that belongs to the backend. The author offers an algorithm of saving data wiht de-duplication, and also

gives the results of comparative tests on the speed of the algorithms of saving and restoring information when working with relational database management systems from different manufacturers.

Keywords: file, data, backup, de-duplication, database

About the authors:
Sergey M. Taranin, orcid.org/0000-0001-8117-7358, PhD,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia,
e-mail: staranin0208@yandex.ru